

XSY
Security Review

Cantina Managed review by:

Kaden, Lead Security Researcher
Rvierdiiev, Security Researcher

September 15, 2025

Contents

1 Introduction **2**
- 1.1 About Cantina . . . 2
- 1.2 Disclaimer . . . 2
- 1.3 Risk assessment . . . 2
  - 1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**

3 Findings **4**
- 3.1 High Risk . . . 4
  - 3.1.1 Locked requests can be redeemed . . . 4
  - 3.1.2 Donation attack on vault depositors . . . 4
- 3.2 Medium Risk . . . 5
  - 3.2.1 Infinite loop due to incrementor pattern . . . 5
- 3.3 Low Risk . . . 6
  - 3.3.1 `requestId == 0` ERC-7540 non-compliance . . . 6
  - 3.3.2 Rounding against vault . . . 6
  - 3.3.3 Incorrect maxMint calculation . . . 7
- 3.4 Informational . . . 7
  - 3.4.1 Incorrect event parameter . . . 7
  - 3.4.2 Users can donate directly bypassing whenNotPaused modifier . . . 8

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

XSY is building a Next Gen Digital Synthetic Dollar, $UTY, creating a structured, scalable way to unlock blockchain ecosystems dormant potential.

From Sep 7th to Sep 11th the Cantina team conducted a review of UTYAsyncVault on commit hash 5cdf901f. The team identified a total of **8** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	2	2	0
Medium Risk	1	1	0
Low Risk	3	3	0
Gas Optimizations	0	0	0
Informational	2	2	0
Total	**8**	**8**	**0**

3 Findings

3.1 High Risk

3.1.1 Locked requests can be redeemed

Severity: High Risk

Context: *(No context files were provided by the reviewer)* **Description:** In `UTYAsyncVaultV1.redeem`, we loop over the `controller`'s requests, redeeming them one by one:

```
// Check user's withdrawal requests one by one and redeem
for (uint256 i; i < length;) {
    if (sharesLeft == 0) {
        break; // No more shares to redeem
    }

    uint256 requestId = requestIdsCopy[i];
    WithdrawalRequest storage request = $.withdrawalRequests[requestId];

    if (sharesLeft >= request.shares) {
        // Full redemption
        assets += request.assets;
        sharesLeft -= request.shares;

        _closeRequest(requestId);
    } else {
        // Partial redemption - rounding errors up to 2 wei are possible compared to full redemption
        uint256 assetsRedeemed = Math.mulDiv(request.assets, sharesLeft, request.shares, Math.Rounding.Floor);
        assets += assetsRedeemed;

        // Update the request with the remaining shares and assets
        request.shares -= sharesLeft;
        request.assets -= assetsRedeemed;
        break;
    }

    // Save 30-40 gas by using unchecked increment
    unchecked {
        ++i;
    }
}
```

However, unlike in `withdraw`, we never validate that the request to redeem is actually unlocked. As a result, it's possible to redeem locked requests.

Note that although we validate that the amount of `shares` to redeem does not exceed `maxShares`, any time a locked request is lower indexed than an unlocked request, we will allow the amount of unlocked shares to be withdrawn from the locked request. It's not safe to assume that requests will be ordered based on unlock times due to the fact that removing requests from the enumerable set uses a 'swap and pop' mechanism whereby we replace the element being removed with the last one in the array and then remove the last element:

```
// To delete an element from the _values array in O(1), we swap the element to delete with the last one in
// the array, and then remove the last element (sometimes called as 'swap and pop').
// This modifies the order of the array, as noted in {at}.
```

Recommendation: For each `request` to redeem, validate `_isUnlocked(request.unlockTime)`, continuing to the next iteration if it is not unlocked.

XSY Labs: Fixed in 9796ebb.

Cantina Managed: Fixed as recommended.

3.1.2 Donation attack on vault depositors

Severity: High Risk

Context: *(No context files were provided by the reviewer)*

Description: The `UTYAsyncVaultV1` contract extends `ERC4626Upgradeable`, which provides the `_decimalsOffset()` hook to help mitigate donation attacks. By default, `_decimalsOffset()` returns `0`, but protocols can override it to increase share granularity and reduce attack viability.

Currently, `UTYAsyncVaultV1` inherits the default implementation without overriding `_decimalsOffset()`. This leaves the vault exposed to donation attacks, where an attacker manipulates the share price in their favor, potentially causing users to mint `0` shares and lose funds.

Proof of Concept:

```
function test_FirstDepositorAttack() public {
    // Attacker makes minimal deposit
    vm.prank(attacker);
    vault.deposit(1, attacker);
    uint attackerShares = vault.balanceOf(attacker);
    console.log("attacker shares: ", attackerShares);

    // Attacker donates large amount to increase share price
    vm.prank(attacker);
    vault.donate(1e18);
    console.log("total assets after donation: ", vault.totalAssets());

    // Legitimate user deposits slightly less than needed for 1 share
    uint assetsTo1Share = vault.previewMint(1);
    console.log("assets to 1 share: ", assetsTo1Share);

    for (uint i = 0; i < 3; i++) {
        depositUser2(assetsTo1Share - 1);
    }
    console.log("total assets after deposits: ", vault.totalAssets());

    // Only attacker holds shares
    console.log("total shares after deposit: ", vault.totalSupply());
    assertEq(vault.totalSupply(), 1);

    // Attacker redeems more than they donated
    uint256 attackerAssetValue = vault.previewRequestRedeem(1);
    assertGt(attackerAssetValue, 1e18 + 1);
    console.log("attacker redeem: ", attackerAssetValue);
}

function depositUser2(uint assets) private {
    vm.prank(user2);
    vault.deposit(assets, user2);
}
```

This test demonstrates how the attacker donates `1e18` assets to inflate share price. Subsequent users who deposit slightly less than the amount needed for 1 share receive 0 shares, effectively losing funds. Meanwhile, the attacker profits.

Recommendation: Override the `_decimalsOffset()` function in `UTYAsyncVaultV1` to increase share precision and make donation-based attacks economically unfeasible. Additionally, consider implementing minimum share minting checks to prevent deposits that would result in `0` shares.

XSY Labs: Fixed in a987342 by preventing deposits with `assets == 0 || shares == 0`.

Cantina Managed: Fix confirmed.

3.2 Medium Risk

3.2.1 Infinite loop due to incrementor pattern

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: In `UTYAsyncVaultV1.withdraw`, we loop over each request and validate that it's unlocked, otherwise we continue to the next iteration:

```
if (!_isUnlocked(request.unlockTime)) {
    continue; // Skip requests that are not yet claimable
}
```

The problem with this logic is that while we `continue` the loop, `i` is only incremented at the end of the loop iteration, after we `continue`:

```
// Save 30-40 gas by using unchecked increment
unchecked {
    ++i;
}
```

As a result, in case we reach a locked `request`, we will `continue` without incrementing `i`, resulting in the same iteration being repeated infinitely.

Recommendation: While this can be fixed by incrementing `i` immediately before `continue` in the `!_isUnlocked(request.unlockTime)` case, as of Solidity v0.8.22, unchecked loop incrementors are automatically generated, so we could replace the use of this pattern altogether with the standard `for (uint256 i; i < length; ++i)` pattern.

XSY Labs: Fixed in a304c0b by removing unchecked incrementor pattern.

Cantina Managed: Fixed as recommended.

3.3 Low Risk

3.3.1 `requestId == 0` ERC-7540 non-compliance

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: When creating a withdrawal request, we set the `requestId` as `$.nextRequestId`, before incrementing it:

```
requestId = $.nextRequestId;
$.nextRequestId += 1;
```

Since the initial value of `$.nextRequestId` is 0, the first withdrawal request will have a `requestId` of 0. This behavior does not comply with the ERC-7540 spec, which notes that, ”If a Vault returns 0 for the requestId of any request, it MUST return 0 for all requests.”.

Recommendation: Use an initial `nextRequestId` value of 1.

XSY Labs: Fixed in 666f162.

Cantina Managed: Fixed as recommended.

3.3.2 Rounding against vault

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: For partial redemptions in `UTYAsyncVaultV1.withdraw`, we compute the amount of `sharesRedeemed`, rounding down the result:

```
uint256 sharesRedeemed = Math.mulDiv(request.shares, assetsLeft, request.assets, Math.Rounding.Floor);
```

As noted in the ERC-4626 spec, we should always round in favor of the vault: ”EIP-4626 Vault implementers should be aware of the need for specific, opposing rounding directions across the different mutable and view methods, as it is considered most secure to favor the Vault itself during calculations over its users:

- If (1) it's calculating how many shares to issue to a user for a certain amount of the underlying tokens they provide or (2) it's determining the amount of the underlying tokens to transfer to them for returning a certain amount of shares, it should round down.
- If (1) it's calculating the amount of shares a user has to supply to receive a given amount of the underlying tokens or (2) it's calculating the amount of underlying tokens a user has to provide to receive a certain amount of shares, it should round up.”.

In this case, since we are calculating the amount of shares a user has to supply to receive a given amount of asset tokens, we should round up the result.

Recommendation: Round up the computation of `sharesRedeemed`:

```
- uint256 sharesRedeemed = Math.mulDiv(request.shares, assetsLeft, request.assets, Math.Rounding.Floor);
+ uint256 sharesRedeemed = Math.mulDiv(request.shares, assetsLeft, request.assets, Math.Rounding.Ceil);
```

XSY Labs: Fixed in 228fc52.

Cantina Managed: Fixed as recommended.

3.3.3 Incorrect maxMint calculation

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: The `maxMint()` function is intended to return the maximum number of shares that a user may mint:

```
uint256 maxShares = maxMint(receiver);
if (shares > maxShares) {
    revert ERC4626ExceededMaxMint(receiver, shares, maxShares);
}
```

However, the current implementation calls `previewMint(maxDeposit(address(user)))` instead of `previewDeposit(maxDeposit(address(user)))`:

```
function maxMint(address user) public view virtual override(IERC4626, ERC4626Upgradeable) returns (uint256) {
    return previewMint(maxDeposit(address(user)));
}
```

This is incorrect because calling `previewMint(maxDeposit(address(user)))` treats the *assets* value as *shares*, producing a logically wrong result. With both vault and underlying token decimals == 18 this mistake may be masked, but if either token uses different decimals the calculation becomes incorrect and `maxMint()` will return an invalid cap - potentially allowing unexpected mint behavior (either too permissive or too restrictive).

Recommendation: Replace the incorrect call with `previewDeposit()` so the function converts the maximum assets into shares correctly.

```
function maxMint(address user) public view virtual override(IERC4626, ERC4626Upgradeable) returns (uint256) {
    return previewDeposit(maxDeposit(address(user)));
}
```

XSY Labs: Fixed in 3f62cc9

Cantina Managed: Fixed as recommended.

3.4 Informational

3.4.1 Incorrect event parameter

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The `Donation` event labels the third parameter as `newSharePrice`:

```
event Donation(address indexed donor, uint256 indexed amount, uint256 indexed newSharePrice);
```

When this parameter is emitted in `UTYAsyncVaultV1.donate`, we pass `previewDeposit(amount)`:

```
emit Donation(sender, amount, previewDeposit(amount));
```

The value provided to the `newSharePrice` parameter here is the amount of shares corresponding to the amount of assets donated, which does not seem consistent with the parameter name. Instead, it's expected that we would provide the assets per share exchange rate to this parameter.

Recommendation: Consider changing the value provided to the `newSharePrice` parameter to be the share price.

XSY Labs: Fixed in 50110c2.

Cantina Managed: Fixed as recommended.

3.4.2 Users can donate directly bypassing whenNotPaused modifier

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The `donate()` function is designed to allow anyone to send assets into the vault, and it applies the `whenNotPaused` modifier to restrict this functionality when the contract is paused.

However, this restriction is ineffective because anyone can simply transfer assets directly to the vault contract address without calling `donate()`. As a result, the pause mechanism does not fully prevent deposits, and the `whenNotPaused` modifier provides a false sense of security.

Recommendation: Consider removing the `whenNotPaused` modifier from `donate()` to avoid misleading assumptions.

XSY Labs: Fixed in 1d35bbc.

Cantina Managed: Fixed as recommended.